1 Overview: fingerprinting

Fingerprinting is a procedure that maps an arbitrarily large data entity to a much smaller bit string, called fingerprint, that can uniquely identify the original data for all practical purposes in the ideal case [Bro93]. This lecture introduces several different fingerprinting algorithms for matrices and strings, and discusses their performance and cost.

2 An intuitive way: fingerprinting with hash function

Suppose Alice has a string \( x \in \mathcal{U} \) that she wants to send to Bob. Bob has another string \( y \in \mathcal{U} \), and wants to know whether \( x = y \). An intuitive fingerprinting algorithm will be picking a hash function \( h : \mathcal{U} \rightarrow [m] \), and sending the string \( x \) in form of \((h(x), h)\). Then Bob will check whether \( \)\( h(y) = h(x) \). This algorithm has false negative rate = 0, and false positive rate = \( \frac{1}{m} \) or \( \frac{1}{m^2} \) for \( h \in \mathcal{H} \) universal / pairwise independent, respectively.

Unfortunately, the algorithm is not practical due to its prohibitively large space cost. For instance, suppose Alice is using the Carter-Wegman hashing: \( h : [U] \rightarrow [B] \) such that \( h(x) := (ax + b) \mod p \) for some \( a, b \in \mathbb{Z}_p \), \( p \geq U \) for pairwise independence \( h \). Then,

- sending the plain string \( x \) takes \( \log U \) bits, while
- sending \((h(x), h_{a,b})\) takes \( \log B + 2 \log p > \log U \) bits.

3 Matrix equality testing

**Problem setting**  Given matrices \( A, B, C \in \mathbb{R}^{n \times n} \) for some large \( n \), we want to know whether \( AB = C \).

**Naive approach**  Check whether \( AB - C = 0 \). Computing \( AB \) takes \( O(n^3) \) FLOPs. It is possible to improve this step to \( O(n^\nu) \) FLOPs for \( \nu = 2.373 \), which is still prohibitively expensive.

**Fingerprinting approach**  Draw a random binary vector \( r \in \{0, 1\}^n \), with i.i.d. entries. Check whether \( ABr = Cr \). This algorithm takes only \( O(n^2) \) FLOPs to evaluate. Its false negative rate = 0, and we will show that its false positive rate \( \leq \frac{1}{2} \).

**Claim 1.** The false positive rate = \( \Pr[(AB - C)r = 0 \cap AB \neq C] \leq \frac{1}{2} \).

**Proof.** We can observe that

\[
\Pr[(AB - C)r = 0 \cap AB \neq C] = \Pr[\text{all non-zero entries in } D := AB - C \text{ cancel each other}]
\leq \Pr[r_j = 1 \mid \sum_{k \neq j} r_k \cdot D(:,k) = -D(:,j)]
= \Pr[r_j = 1] = \frac{1}{2}
\]

\( \square \)
Following the same reasoning, we can show that the false positive rate can be improved to be $\leq \frac{1}{k}$ by drawing $r \in [k]^n$ i.i.d. instead.

4 Polynomial identity testing

**Problem** Given $P(x)$, $Q(x)$, $R(x) \in R[x]$ polynomials of degree $d$, $d$, $2d$, respectively. We ask whether $P(x)Q(x) = R(x)$. More generally, we ask whether $P(x) = Q(x)$, $\deg(P) = \deg(Q) = d$.

**Naive approach** We can observe that the polynomial $P(x) - Q(x)$ is of degree at most $d$, and therefore has at most $d$ roots. By picking random elements $x \in [O(d)]$, we can check whether $P(x) - Q(x) = 0$. This method has 0 false negative rate and constant false positive rate $\frac{1}{c}$ when picking $x \in [cd]$. However, the direct evaluation of $P(x)$ and $Q(x)$ involves storing numbers as large as $O(d^d)$ which takes $O(d \log d)$ space.

**Fingerprinting** We can improve the space cost by projecting $P(x) - Q(x)$ into a finite field $\mathbb{F}_p$ for some prime $p \geq 2d$. Then the polynomial identity test on random $x \in \mathbb{F}_p^*$ has false positive rate $= \frac{d}{p} \leq \frac{1}{2}$. In addition, the Schwartz-Zippel Lemma [Sch80; Zip79] suggests that the same holds for multivariate polynomials. That is, for $\deg(P(x_1, x_2, \ldots)) = d$, let $p \geq 2d$. Choosing $x_1, x_2, \cdots \in \mathbb{F}_p$ randomly, we have $\Pr[P(x_1, x_2, \ldots) = 0 \mid P(x) \neq 0] \leq \frac{d}{p}$.

5 String testing

Suppose Alice has string $a = a_0, \ldots, a_{n-1} \in \{0, 1\}^n$, and Bob has string $b = b_0, \ldots, b_{n-1} \in \{0, 1\}^n$. Suppose they would like to test whether $a = b$ by fingerprinting. Alice then needs to send her choice of hash function $h$ and $h(a)$ so that Bob can check whether $h(a) = h(b)$.

How large does Alice’s message $(h(a), h)$ need to be?

5.1 String testing via Rabin-Karp hashing

We can test whether $a = b$ by treating $\{a_i\}_{i=0}^{n-1}, \{b_i\}_{i=0}^{n-1}$ as coefficients of polynomials.

Using the techniques described for polynomial identity testing, we can fix $p > 2n$ and choose $x \in \{0, \ldots, p - 1\}$ at random and evaluate

$$h(a) = \sum_{i=0}^{n-1} a_i x^i \mod p = \sum_{i=0}^{n-1} b_i x^i \mod p = h(b).$$

This test has a false positive rate of at most $\frac{2}{p}$, since there is at most this chance of randomly choosing $x \in \{0, \ldots, p - 1\}$ as one of the $n$ roots of $\sum_{i=1}^{n-1} a_i x^i$. Taking $p > n^2$ gives failure rate at most $\frac{1}{2}$.

For Alice and Bob to do their fingerprinting test, Alice must send only $O(\log p) \sim O(\log n)$ bits for $h(a)$ and her random choice of $x \in \{0, \ldots, p - 1\}$.

5.1.1 An application of Rabin-Karp Hashing: Pattern Matching

Consider the pattern matching problem, where we are given two strings $a = a_1, \ldots, a_m$ and $b = b_1, \ldots, b_n$ for $m < n$, and we would like to find all indices $i$ such that $a_1, \ldots, a_m = b_i, \ldots, b_{i+m-1}$, i.e. find all the locations where $a$ occurs as a substring in $b$. Naively, there is an $O(mn)$ algorithm that solves this problem. There is a deterministic $O(m + n) \sim O(n)$ algorithm that solves this problem (KMP algorithm), but it is complicated! We can solve this problem in $O(n)$ time with high probability using Rabin-Karp hashing.

We choose $h$ to be

$$h(a) = \sum_{j=1}^{m} a_j x^{m-j} \mod p.$$
Then, given $h(b_1, \ldots, b_m)$, there is a simple formula for evaluating $h(b_2, \ldots, h_{m+2})$:

$$h(b_2, \ldots, h_{m+2}) = \sum_{j=2}^{m+1} b_j x^{m+1-j} \mod p = x h(b_1, \ldots, b_m) + b_{m+1} - b_1 x^{m+1} \mod p.$$ 

More generally, given $h(a)$ and $h(b_{i-1}, \ldots, b_{i-1+m})$ for $i \in \{2, \ldots, n - m\}$, we can compute $h(b_i, \ldots, b_{i+m})$ in $O(1)$ time. This gives an $O(m + n)$ algorithm for the pattern matching problem.

The expected number of false matches is at most $n \cdot \frac{m}{p}$, using the union bound over all length $m$ substrings of $b$. Taking $p > n^3$ gives failure rate at most $1/n$.

Above is a Monte-Carlo algorithm that runs in $O(m + n)$ time. We can make it a Las Vegas algorithm by doing an exhaustive check on each substring of $b$ that tests positively as matching $a$. This would take $O(n + am)$ time, where $\alpha$ is the number of occurrences of $a$ in $b$.

5.2 String testing via primality testing

Consider again the problem of testing equality of strings $a = a_0, \ldots, a_{n-1} \in \{0, 1\}^n$ and $b = b_0, \ldots, b_{n-1} \in \{0, 1\}^n$ using fingerprinting techniques. Using Rabin-Karp, we chose $h(a) = \sum_{i=0}^{n-1} a_i x^i \mod p$ for $p > n$ fixed and $x \in \mathbb{F}_p$ chosen at random.

Alternatively, we can fix $x$ and choose $p$ at random. This is analogous to treating $a$ and $b$ as bit strings that represent integers $\sum_i a_i 2^i$, $\sum_i b_i 2^i$.

Clearly, the false positive rate is 0, but for $a \neq b$, what is the probability that $a - b \equiv 0 \mod p$ for a random choice of $p$? We have a false positive iff $p|(a - b)$, where $(a - b)$ is an $(n + 1)$ bit number. Since $a - b \leq 2^{n+1}$, it has at most $O(n)$ prime factors, as a prime factor is at least 2.

The density of prime numbers is $O(1/\log n)$. Therefore, if we choose $p$ a random prime such that $1 \leq p \leq O(n^2 \log n)$, then there will be at most $n^2$ primes to choose from. Only $n$ of them happen to divide $(a - b)$. This gives false positive rate at most $1/n$. For Alice and Bob to do their fingerprinting test, Alice only needs $O(\log k) \sim O(\log n)$ bits to share her fingerprint and her random choice of prime $p$.

How do we choose a random prime? If we can primality test, we can do $O(\log n)$ queries until we find one, with high probability. The Agarwal-Biswas algorithm uses the fact that for $N$ a positive integer

$$(z^N + 1) \equiv (z + 1)^N \mod N \iff N \text{ is prime}.$$ 

Instead of evaluating the above (mod $N$), they evaluate it (mod $Q$) for some random choice of $Q$. The details are outside the scope of this lecture. The primality test succeeds with high probability and can be executed in polylog time.

References

